Our Ref. No. 042390.P8629X
Express Mail No.: EL466330900US

UNITED STATES PATENT APPLICATION

FOR

ATTESTATION KEY MEMORY DEVICE AND BUS

INVENTORS:

Carl M. Ellison
Roger A. Golliver
Howard C. Herbert
Derrick C. Lin
Francis X. McKeen
Gilbert Neiger
Ken Reneris
James A. Sutton
Shreekant S. Thakkar
Milland Mittal

PREPARED BY:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 Wilshire Blvd., 7th Floor
Los Angeles, CA 90025-1026
(714) 557-3800

ATTESTATION KEY MEMORY DEVICE AND BUS

RELATED APPLICATIONS

This application is a continuation-in-part of application 09/541,687 filed March 31, 2000.

BACKGROUND

1. Field of the Invention

This invention relates to microprocessors. In particular, the invention relates to processor security.

2. Description of Related Art

Advances in microprocessor and communication technologies have opened up many opportunities for applications that go beyond the traditional ways of doing business. Electronic commerce (E-commerce) and business-to-business (B2B) transactions are now becoming popular, reaching the global markets at a fast rate. Unfortunately, while modern microprocessor systems provide users convenient and efficient methods of doing business, communicating and transacting, they are also vulnerable to unscrupulous attacks. Examples of these attacks include virus, intrusion, security breach, and tampering, to name a few. Computer security, therefore, is becoming more and more important to protect the integrity of the computer systems and increase the trust of users.

Threats caused by unscrupulous attacks may be in a number of forms. Attacks may be remote without requiring physical accesses. An invasive remote-launched attack by hackers may disrupt the normal operation of a system connected to thousands or even millions of users. A virus program may corrupt code and/or data of a single-user platform.

Existing techniques to protect against attacks have a number of drawbacks. Anti-virus programs can only scan and detect known viruses.

Most anti-virus programs use a weak policy in which a file or program is assumed good until proved bad. For many security applications, this weak policy may not be appropriate. In addition, most anti-virus programs are used locally where they are resident in the platform. This may not be suitable in a group work environment. Security co-processors or smart cards using cryptographic or other security techniques have limitations in speed performance, memory capacity, and flexibility. Redesigning operating systems creates software compatibility issues and causes tremendous investment in development efforts.

BRIEF DESCRIPTION OF THE DRAWINGS

The features and advantages of the present invention will become apparent from the following detailed description of the present invention in which:

Figure 1A is a diagram illustrating a logical architecture according to one embodiment of the invention.

Figure 1B is a diagram illustrating accessibility of various elements in the operating system and the processor according to one embodiment of the invention.

Figure 1C is a diagram illustrating a computer system in which one embodiment of the invention can be practiced.

Figure 2 is a diagram illustrating the token bus interface shown in Figure 1C according to one embodiment of the invention.

Figure 3 is a diagram illustrating the configuration storage shown in Figure 2 according to one embodiment of the invention.

Figure 4 is a diagram illustrating the signing operation shown in Figure 3 according to one embodiment of the invention.

Figure 5 is a diagram illustrating the status register shown in Figure 3 according to one embodiment of the invention.

DETAILED DESCRIPTION

In the following description, for purposes of explanation, numerous details are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that these specific details are not required in order to practice the present invention. In other instances, well-known electrical structures and circuits are shown in block diagram form in order not to obscure the present invention.

ARCHITECTURE OVERVIEW

One principle for providing security in a computer system or platform is the concept of an isolated execution architecture. The isolated execution architecture includes logical and physical definitions of hardware and software components that interact directly or indirectly with an operating system of the computer system or platform. An operating system and the processor may have several levels of hierarchy, referred to as rings, corresponding to various operational modes. A ring is a logical division of hardware and software components that are designed to perform dedicated tasks within the operating system. The division is typically based on the degree or level of privilege, namely, the ability to make changes to the platform. For example, a ring-0 is the innermost ring, being at the highest level of the hierarchy. Ring-0 encompasses the most critical, privileged components. In addition, modules in Ring-0 can also access to lesser privileged data, but not vice versa. Ring-3 is the outermost ring, being at the lowest level of the hierarchy. Ring-3 typically encompasses users or applications level and executes the least trusted code. It is noted that the level of the ring hierarchy is independent to the level of the security protection of that ring.

Figure 1A is a diagram illustrating a logical operating architecture 50 according to one embodiment of the invention. The logical operating

architecture 50 is an abstraction of the components of an operating system and the processor. The logical operating architecture 50 includes ring-0 10, ring-1 20, ring-2 30, ring-3 40, and a processor nub loader 52. The processor nub loader 52 is an instance of a processor executive (PE) handler. The PE handler is used to handle and/or manage a processor executive (PE) as will be discussed later. The logical operating architecture 50 has two modes of operation: normal execution mode and isolated execution mode. Each ring in the logical operating architecture 50 can operate in both modes. The processor nub loader 52 operates only in the isolated execution mode.

Ring-0 10 includes two portions: a normal execution Ring-0 11 and an isolated execution Ring-0 15. The normal execution Ring-0 11 includes software modules that are critical for the operating system, usually referred to as kernel. These software modules include primary operating system (e.g., kernel) 12, software drivers 13, and hardware drivers 14. The isolated execution Ring-0 15 includes an operating system (OS) nub 16 and a processor nub 18. The OS nub 16 and the processor nub 18 are instances of an OS executive (OSE) and processor executive (PE), respectively. The OSE and the PE are part of executive entities that operate in a secure environment associated with the isolated area 70 and the isolated execution mode. The processor nub loader 52 is a protected bootstrap loader code held within a chipset in the system and is responsible for loading the processor nub 18 from the processor or chipset into an isolated area as will be explained later.

Similarly, ring-1 20, ring-2 30, and ring-3 40 include normal execution ring-1 21, ring-2 31, ring-3 41, and isolated execution ring-1 25, ring-2 35, and ring-3 45, respectively. In particular, normal execution ring-3 includes N applications 42_1 to 42_N and isolated execution ring-3 includes K applets 46_1 to 46_K.

One concept of the isolated execution architecture is the creation of an isolated region in the system memory, referred to as an isolated area, which is protected by both the processor and chipset in the computer system. Portions of the isolated region may also be in cache memory. Access to this isolated region is permitted only from a front side bus (FSB) of the processor, using special bus (e.g., memory read and write) cycles, referred to as isolated read and write cycles. The special bus cycles are also used for snooping. The isolated read and write cycles are issued by the processor executing in an isolated execution mode when accessing the isolated area. The isolated execution mode is initialized using a privileged instruction in the processor, combined with the processor nub loader 52. The processor nub loader 52 verifies and loads a ring-0 nub software module (e.g., processor nub 18) into the isolated area. The processor nub 18 provides hardware-related services for the isolated execution.

One task of the processor nub loader 52 and processor nub 18 is to verify and load the ring-0 OS nub 16 into the isolated area, and to generate the root of a key hierarchy unique to a combination of the platform, the processor nub 18, and the operating system nub 16. The operating system nub 16 provides links to services in the primary OS 12 (e.g., the unprotected operating system), provides page management within the isolated area, and has the responsibility for loading ring-3 application modules 45, including applets 46_1 to 46_K, into protected pages allocated in the isolated area. The operating system nub 16 may also load ring-0 supporting modules.

The operating system nub 16 may choose to support paging of data between the isolated area and ordinary (e.g., non-isolated) memory. If so, then the operating system nub 16 is also responsible for encrypting and hashing the isolated area pages before evicting the page to the ordinary memory, and for checking the page contents upon restoration of the page.

The isolated mode applets 46_1 to 46_K and their data are tamper-resistant and monitor-resistant from all software attacks from other applets, as well as from non-isolated-space applications (e.g., 42_1 to 42_N), drivers and even the primary operating system 12. The software that can interfere with or monitor the applet's execution is the processor nub loader 52, processor nub 18 or the operating system nub 16.

Figure 1B is a diagram illustrating accessibility of various elements in the operating system 10 and the processor according to one embodiment of the invention. For illustration purposes, only elements of ring-0 10 and ring-3 40 are shown. The various elements in the logical operating architecture 50 access an accessible physical memory 60 according to their ring hierarchy and the execution mode.

The accessible physical memory 60 includes an isolated area 70 and a non-isolated area 80. The isolated area 70 includes applet pages 72 and nub pages 74. The non-isolated area 80 includes application pages 82 and operating system pages 84. The isolated area 70 is accessible only to elements of the operating system and processor operating in isolated execution mode. The non-isolated area 80 is accessible to all elements of the ring-0 operating system and to the processor.

The normal execution ring-0 11 including the primary OS 12, the software drivers 13, and the hardware drivers 14, can access both the OS pages 84 and the application pages 82. The normal execution ring-3, including applications 42_1 to 42_N, can access only to the application pages 82. Generally applications can only access to their own pages, however, the OS typically provides services for sharing memory in controlled methods. Both the normal execution ring-0 11 and ring-3 41, however, cannot access the isolated area 70.

The isolated execution ring-0 15, including the OS nub 16 and the processor nub 18, can access to both of the isolated area 70, including the applet pages 72 and the nub pages 74, and the non-isolated area 80, including the application pages 82 and the OS pages 84. The isolated execution ring-3 45, including applets 46_1 to 46_K, can access only applet pages 72. The applets 46_1 to 46_K reside in the isolated area 70. In general, applets can only access their own pages; however, the OS nub 16 can also provides services for the applet to share memory (e.g., share memory with other applets or with non-isolated area applications).

Figure 1C is a diagram illustrating a computer system 100 in which one embodiment of the invention can be practiced. The computer system 100 includes a processor 110, a host bus 120, a memory controller hub (MCH) 130, a system memory 140, an input/output controller hub (ICH) 150, a non-volatile memory, or system flash, 160, a mass storage device 170, input/output devices 175, a token bus 180, a motherboard (MB) token 182, a reader 184, and a token 186. The MCH 130 may be integrated into a chipset that integrates multiple functionalities such as the isolated execution mode, host-to-peripheral bus interface, memory control. Similarly, the ICH 150 may also be integrated into a chipset together or separate from the MCH 130 to perform I/O functions. For clarity, not all the peripheral buses are shown. It is contemplated that the system 100 may also include peripheral buses such as Peripheral Component Interconnect (PCI), accelerated graphics port (AGP), Industry Standard Architecture (ISA) bus, and Universal Serial Bus (USB), etc. The "token bus" may be part of the USB bus, e.g., it may be hosted on the USB bus.

The processor 110 represents a central processing unit of any type of architecture, such as complex instruction set computers (CISC), reduced instruction set computers (RISC), very long instruction word (VLIW), or hybrid

architecture. In one embodiment, the processor 110 is compatible with an Intel Architecture (IA) processor, such as the Pentium™ series, the IA-32™ and the IA-64™. The processor 110 includes a normal execution mode 112 and an isolated execution circuit 115. The normal execution mode 112 is the mode in which the processor 110 operates in a non-secure environment, or a normal environment without the security features provided by the isolated execution mode. The isolated execution circuit 115 provides a mechanism to allow the processor 110 to operate in an isolated execution mode. The isolated execution circuit 115 provides hardware and software support for the isolated execution mode. This support includes configuration for isolated execution, definition of an isolated area, definition (e.g., decoding and execution) of isolated instructions, generation of isolated access bus cycles, and access checking.

In one embodiment, the computer system 100 can be a single processor system, such as a desktop computer, which has only one main central processing unit, e.g. processor 110. In other embodiments, the computer system 100 can include multiple processors, e.g. processors 110, 110a, 110b, etc., as shown in Figure 1C. Thus, the computer system 100 can be a multi-processor computer system having any number of processors. For example, the multi-processor computer system 100 can operate as part of a server or workstation environment. The basic description and operation of processor 110 will be discussed in detail below. It will be appreciated by those skilled in the art that the basic description and operation of processor 110 applies to the other processors 110a and 110b, shown in Figure 1C, as well as any number of other processors that may be utilized in the multi-processor computer system 100 according to one embodiment of the present invention.

The processor 110 may also have multiple logical processors. A logical processor, sometimes referred to as a thread, is a functional unit within a physical processor having an architectural state and physical resources allocated according to some partitioning policy. Within the context of the present invention, the terms "thread" and "logical processor" are used to mean the same thing. A multi-threaded processor is a processor having multiple threads or multiple logical processors. A multi-processor system (e.g., the system comprising the processors 110, 110a, and 110b) may have multiple multi-threaded processors.

The host bus 120 provides interface signals to allow the processor 110 or processors 110, 100a, and 110b to communicate with other processors or devices, e.g., the MCH 130. In addition to normal mode, the host bus 120 provides an isolated access bus mode with corresponding interface signals for memory read and write cycles. The isolated access bus mode is asserted on memory accesses initiated while the processor 110 is in the isolated execution mode and it is accessing memory within the isolated area. The isolated access bus mode is also asserted on instruction pre-fetch and cache write-back cycles if the address is within the isolated area address range. The isolated access bus mode is configured within the processor 110. The processor 110 responds to a snoop cycle to a cached address when the isolated access bus mode on the FSB matches the mode of the cached address.

The MCH 130 provides control and configuration of system memory 140. The MCH 130 provides interface circuits to recognize and service isolated access assertions on memory reference bus cycles, including isolated memory read and write cycles. In addition, the MCH 130 has memory range registers (e.g., base and length registers) to represent the isolated area in the system memory 140. Once configured, the MCH 130

aborts any access to the isolated area that does not have the isolated access bus mode asserted.

The system memory 140 stores system code and data. The system memory 140 is typically implemented with dynamic random access memory (DRAM) or static random access memory (SRAM). The system memory 140 includes the accessible physical memory 60 (shown in Figure 1B). The accessible physical memory includes a loaded operating system 142, the isolated area 70 (shown in Figure 1B), and an isolated control and status space 148. The loaded operating system 142 is the portion of the operating system that is loaded into the system memory 140. The loaded OS 142 is typically loaded from a mass storage device via some boot code in a boot storage such as a boot read only memory (ROM). The isolated area 70, as shown in Figure 1B, is the memory area that is defined by the processor 110 when operating in the isolated execution mode. Access to the isolated area 70 is restricted and is enforced by the processor 110 and/or the MCH 130 or other chipset that integrates the isolated area functionalities. The isolated control and status space 148 is an input/output (I/O)-like, independent address space defined by the processor 110. The isolated control and status space 148 contains mainly the isolated execution control and status registers. The isolated control and status space 148 does not overlap any existing address space and is accessed using the isolated bus cycles. The system memory 140 may also include other programs or data that are not shown.

The ICH 150 represents a known single point in the system having the isolated execution functionality. For clarity, only one ICH 150 is shown. The system 100 may have many ICH's similar to the ICH 150. When there are multiple ICH's, a designated ICH is selected to control the isolated area configuration and status. In one embodiment, this selection is performed by an external strapping pin. As is known by one skilled in the art, other

methods of selecting can be used, including using programmable configuring registers. The ICH 150 has a number of functionalities that are designed to support the isolated execution mode in addition to the traditional I/O functions. In particular, the ICH 150 includes an isolated bus cycle interface 152, the processor nub loader 52 (shown in Figure 1A), a digest memory 154, a cryptographic key storage 155, an isolated execution logical processor manager 156, and a token bus interface 159.

The isolated bus cycle interface 152 includes circuitry to interface to the isolated bus cycle signals to recognize and service isolated bus cycles, such as the isolated read and write bus cycles. The processor nub loader 52, as shown in Figure 1A, includes a processor nub loader code and its digest (e.g., cryptographic hash) value. The processor nub loader 52 is invoked by execution of an appropriate isolated instruction (e.g., Iso_Init) and is transferred to the isolated area 70. From the isolated area 80, the processor nub loader 52 copies the processor nub 18 from the system flash memory (e.g., the processor nub code 18 in non-volatile memory 160) into the isolated area 70, verifies and logs its integrity, and manages a symmetric key used to protect the processor nub's secrets. In one embodiment, the processor nub loader 52 is implemented in read only memory (ROM). For security purposes, the processor nub loader 52 is unchanging, tamper-resistant and non-substitutable. The digest memory 154, typically implemented in RAM, stores the digest (e.g., cryptographic hash) values of the loaded processor nub 18, the operating system nub 16, and any other supervisory modules (e.g., ring-0 modules) loaded into the isolated execution space. The cryptographic key storage 155 holds a symmetric encryption/decryption key that is unique for the platform of the system 100. In one embodiment, the cryptographic key storage 155 includes internal fuses that are programmed at manufacturing. Alternatively, the cryptographic key storage 155 may also be created during manufacturing with a cryptographic random number generator. The isolated

execution logical processor manager 156 manages the operation of logical processors configuring their isolated execution mode support. In one embodiment, the isolated execution logical processor manager 156 includes a logical processor count register that tracks the number of logical processors participating in the isolated execution mode. The token bus interface 159 interfaces to the token bus 180. A combination of the processor nub loader digest, the processor nub digest, the operating system nub digest, and optionally additional digests, represents the overall isolated execution digest, referred to as isolated digest. The isolated digest is a fingerprint identifying the all supervisory code involved in controlling the isolated execution configuration and operation. The isolated digest is used to attest the state of the current isolated execution and to prove the validity of the software loaded into the isolated area..

The non-volatile memory 160 stores non-volatile information. Typically, the non-volatile memory 160 is implemented in flash memory. In one embodiment, the non-volatile memory 160 includes the processor nub 18. The processor nub 18 provides set-up and low-level management of the isolated area 70 (in the system memory 140), including verification, loading, and logging of the operating system nub 16, and the management of the symmetric key used to protect the operating system nub's secrets. The processor nub loader 52 performs some part of the setup and manages/updates the symmetric key before the processor nub 18 and the OS nub 16 are loaded. The processor nub 18 The processor nub 18 may also provide interface abstractions to low-level security services provided by other hardware. The processor nub 18 may also be distributed by the original equipment manufacturer (OEM) or operating system vendor (OSV).

The mass storage device 170 stores archive information such as code (e.g., processor nub 18), programs, files, data, applications (e.g., applications

42_1 to 42_N), applets (e.g., applets 46_1 to 46_K) and operating systems. The mass storage device 170 may include compact disk (CD) ROM 172, floppy diskettes 174, and hard drive 176, and any other storage devices. The mass storage device 170 provides a mechanism to read machine-readable media. When implemented in software, the elements of the present invention are the code segments to perform the necessary tasks. The program or code segments can be stored in a processor readable medium or transmitted by a computer data signal embodied in a carrier wave, or a signal modulated by a carrier, over a transmission medium. The "processor readable medium" may include any medium that can store or transfer information. Examples of the processor readable medium include an electronic circuit, a semiconductor memory device, a ROM, a flash memory, an erasable programmable ROM (EPROM), a floppy diskette, a compact disk CD-ROM, an optical disk, a hard disk, a fiber optical medium, a radio frequency (RF) link, etc. The computer data signal may include any signal that can propagate over a transmission medium such as electronic network channels, optical fibers, air, electromagnetic, RF links, etc. The code segments may be downloaded via computer networks such as the Internet, an Intranet, etc.

I/O devices 175 may include any I/O devices to perform I/O functions. Examples of I/O devices 175 include a controller for input devices (e.g., keyboard, mouse, trackball, pointing device), media card (e.g., audio, video, graphics), a network card, and any other peripheral controllers.

The token bus 180 provides an interface between the ICH 150 and various tokens in the system. A token is a device that performs dedicated input/output functions with security functionalities. A token has characteristics similar to a smart card, including at least one reserved-purpose public/private key pair and the ability to sign data with the private key. Examples of tokens connected to the token bus 180 include a motherboard token 182, a token

reader 184, and other portable tokens 186 (e.g., smart card). The token bus interface 159 in the ICH 150 connects through the token bus 180 to the ICH 150 and ensures that when commanded to prove the state of the isolated execution, the corresponding token (e.g., the motherboard token 182, the token 186) signs only valid isolated digest information. For purposes of security, the token should be connected to the digest memory via the token bus 180.

ATTESTATION KEY MEMORY (AKM) DEVICE AND BUS

In an embodiment of the present invention, a technique is provided for remote attestation. The remote attestation is performed by a device operating in a remote manner with respect to the MCH 130 and the ICH 150 (Figure 1C). Examples of this device include one of the tokens 186. This device is referred to as an attestation key memory (AKM) device. This remote attestation is performed by using a public-private key pair to attest that the isolated execution mode is running with a particular software configuration. Depending on the need of the software utilizing the attestation, the results can be bonded to the platform embodying the secure environment such that future attestation is not required unless there is a significant change in the software configuration. The AKM device contains one or more key pair and may be inserted into the platform by the end user needed to perform the attestation.

The AKM device allows a user to validate the integrity of the isolated area. It able to use the hardware to validate the state of the software. The AKM device provide a simple model for users to understand when there are privacy and anonymity issues. In addition, the AKM device offers some advantages and benefits over a non-pluggable device approach. It also prevents spoofing by emulation software. The AKM device remotely attests

by getting the state of the software broadcast to the remote server. This is done by the NUB and some network interface.

The benefits of using an AKM device includes: distribution of the private key, replacement or removal of the private key if desired, usage of more than one key if desired, remote verification of software on an unknown machine by a remote server, provision of value-added features via the interface bus (e.g., the token bus 180 shown in Figure 1C). The AKM device may be removed or fixed on the motherboard.

In an embodiment of the present invention, an interface maps a device (e.g., the AKM device) via a bus (e.g., the token bus 180 shown in Figure 1C) to an address space of a chipset (e.g., the ICH 150 shown in Figure 1C) in a secure environment for an isolated execution mode. The secure environment is associated with an isolated memory area accessible by at least one processor. A communication storage corresponding to the address space allows the device to exchange security information with the at least one processor in the isolated execution mode in a remote attestation.

Figure 2 is a diagram illustrating the token bus interface 159 shown in Figure 1C according to one embodiment of the invention. The token bus interface 159 includes an interface 210, a communication storage 220, and a chipset storage 270.

The interface 210 provides an interface between an external device (e.g., the tokens 186 shown in Figure 1C) coupled to the token bus 180 (Figure 1C and the chipset (e.g., the ICH 150). The interface 210 includes a decoder 212. The decoder 212 decodes the address space onto the bus 180 so that an access to the chipset is passed to the external device. Typically the address space is a subset of the address space of the chipset 150. In

addition, the decoder 212 allows the device 186 to access the chipset storage 270.

The communication storage 220 is mapped to the address space and allows the device 186 to exchange security information with the chipset 150 or the processor 110. The communication storage 220 includes a configuration storage 230, a status register 240, a command register 250, and an input/output block (IOB) 260. The configuration storage 230 stores configuration information 232. The status register 240 stores device status 242. The command register 250 stores device command 252. The IOB 260 stored input data 262 and output data 264.

The chipset storage 270 stores chipset information such as the system digest in the digest memory 154 (Figure 1C). In particular, the chipset storage 270 includes a processor nub loader hash 272, a chipset hash log 274, a software hash 276, and a nonce 278. The processor nub loader hash 272 and the chipset hash log 274 can be read directly by the AKM device 186 and cannot be intercepted by the running software. The software hash 276 and the nonce 278 are provided by the processor nub 18 (Figure 1A). Furthermore, additional hash registers may be provided for other software hashes.

Figure 3 is a diagram illustrating the configuration storage 230 shown in Figure 2 according to one embodiment of the invention. The configuration storage 230 includes a manufacturer identifier 310, a revision identifier 320, an interface set identifier 330, a static public key 340, and a static key certificate 350. The configuration storage includes a plurality of sub-storages (e.g., public key storage, key certificate storage, interface set storage, revision storage). Typically, the configuration storage 230 is read-only. This device can be attached to the bus 180 and made removable. A removable device is important for proving a platform.

The manufacturer identifier 310 identifies the manufacturer of the AKM device 186. The revision identifier 320 provides a revision number of the AKM device 186. The interface set identifier 330 identifies the interface set that is supported by the device 186. The static public key 340 is a public key with a short key identification. The key certificate 350 is a key certificate with a short key identification.

The interface set identified by the interface set identifier 330 identifies may include an initialization set 360, an attestation set 370, and a device interface set 380. For a typical remote attestation, the initialization set 360 is needed. The initialization set 360 may be hardcoded and is used to reset and initialize the device. The initialization set 360 includes an idle state 362, a reset command 364, a connect command 366, and a reserved operation 368. The idle state 362 indicates that the device is not performing any meaningful operation and is idle. The reset command 364 causes the device to reset and perform a self-test operation. The connect command 366 sets the connect bit in the status register 240. The reserved operation 368 is to be reserved for other operations or commands or for non-implemented operation. A command that corresponds to the reserved operation 368 results in a "not-supported" error.

The attestation set 370 includes a signing operation 372, a public key enumeration 374, and a key certificate enumeration 376. The signing operation 372 provides the remote attestation to verify the validity of the platform running a particular software in the secure environment. The public key enumeration 374 enumerates any additional public keys that are not part of the static configuration information 232 (shown in Figure 2). The key certificate enumeration 376 enumerates any additional key certificates that are not part of the static configuration information 232 (shown in Figure 2).

The device interface set 380 is any additional interface set that can be supported by the AKM device in addition to the initialization set 360 and the attestation set 370.

When the device receives a command, it performs the operation as specified. During this time, the device may update the status register to report any conditions. When the operation is completed, the device writes the result in the IOB 260, clears a time estimate in the status register (discussed below), and clears the command register. When the host processor 110 polls the command register, a zero value indicates the device is idle. The processor 110 then can check the status register 240 for any device fatal error. If there is no fatal error, the host then reads the results from the IOB 260. Alternatively, the device can read the registers and decide whether or not the platform is safe.

Figure 4 is a diagram illustrating the signing operation 372 shown in Figure 3 according to one embodiment of the invention. The signing operation 372 includes a hash function 410 and a cryptographic function 420.

The hash function 410 performs hashing on the processor nub loader hash 272, the chipset hash log 274, the software hash 276, and the nonce 278. The result of this hashing operation is then encrypted by the cryptographic function 420 using the private key 280 stored in the chipset. The result of the encryption becomes the output data 264 to be stored in the IOB 260. When the signing operation 372 is complete, the processor nub 18 retrieves the result from the IOB 260.

Figure 5 is a diagram illustrating the status register shown in Figure 3 according to one embodiment of the invention. The status register 240 includes a self-test field 510, a connection field 520, an estimate field 530, and a reserved field 540.

The self-test field 510 provides a result of the self-test operation in response to the reset command. The result may include a failure. When there is a failure, all results from the device are ignored. This failure code is typically reset by a reset command or a system reset. The connection field 520 indicates that the device is responsive to the connect command. The estimate field 530 provides an estimate in some time unit (e.g., milliseconds) to indicate how long a current operation is expected to take. For example, a value zero indicates that it is less the a millisecond to complete. The reserved field 540 is reserved for future use.

While this invention has been described with reference to illustrative embodiments, this description is not intended to be construed in a limiting sense. Various modifications of the illustrative embodiments, as well as other embodiments of the invention, which are apparent to persons skilled in the art to which the invention pertains are deemed to lie within the spirit and scope of the invention.